Algorytmy i struktury danych.
Laboratorium 3

Dr hab. Bożena Woźna-Szcześniak, prof.UJD

1. Wzorując się wykładem 2 oraz stosując notację $\sim$ oszacuj czas pracy poniższych fragmentów kodu jako funkcję, która przyjmuje dane wejściowe o rozmiarze N. Udokumentuj sposób oszacowania. Dokument przygotuj przy pomocy Latex-a.

   - ```
     int sum = 0;
     for (int n = N; n > 0; n /= 2)
        for (int i = 0; i < n; i++)
           sum++;
     ```
   - ```
     int sum = 0;
     for (int i = 1; i < N; i *= 2)
        for(int j = 0; j < i; j++)
           sum++;
     ```
   - ```
     int sum = 0;
     for (int i = 1; i < N; i *= 2)
        for (int j = 0; j < N; j++)
           sum++;
     ```

2. Opracuj i zaimplementuj rozwiązanie *brute-force* dla problemu 4-SUM:

   > Dane jest zbiór (tablica) złożony z N liczb całkowitych. Problem 4-SUM pyta, czy istnieją elementy a, b, c i c z tego zbioru, takie że $a + b + c + d = 0$, a jeśli tak to ile ich jest?

   - Oszacuj czas działania swojego rozwiązania problemu 4-SUM jako funkcję, która przyjmuje dane wejściowe o rozmiarze N.

3. Lokalne minimum tablicy. Napisz program, który mając tablicę A złożoną z N różnych liczb całkowitych, znajdzie lokalne minimum: indeks i takie, że zarówno $a[i] < a[i-1]$ i $a[i] < a[i+1]$ (zakładając że sąsiednia pozycja znajduje się w granicach tablicy). Na tablicę A nakładamy następujące ograniczenia:

   - Tablice mają co najmniej trzy elementy
   - Pierwsze dwie liczby maleją, a dwie ostatnie rosną.
   - Liczby są rózne

   Program powinien użyć $2log(N)$ porównań w najgorszym przypadku.
     ```

Podpowiedź: Sprawdź wartość środkową w $[N/2]$ i jego dwóch sąsiadów w $[N/2-1]$ i $[N/2+1]$. Jeśli $[N/2]$ jest lokalnym minimum, stop; w przeciwnym wypadku szukaj w połowie mniejszego sąsiada.

Przykład 1: Niech $A = \{8, 5, 7, 2, 3, 4, 1, 9\}$. Tablica A ma kilka lokalnych minimum. Są to: 5, 2 i 1. Minimum, które powinien zwrócić program to 2.

Przykład 2: Niech $A = \{8, 5, 2, 7, 3, 4, 1, 9\}$. Oznaczmy przez L Lewy koniec tablicy (tj. indeks 0), przez P prawy konie koniec tablicy (tj. indeks 7), a przez M środek tablicy.

- $L = 0$; $R = 7$; $M = (L + R)/2 = 3$. Element $A[3] = 7$ NIE jest lokalnym minimum. Wyszukujemy zatem we fragmencie $A[0 \ldots 2]$ odpowiadającym mniejszemu sąsiadowi, tj. 2.
- $L = 0$; $R = 2$; $M = (L + R)/2 = 1$. Element $A[1] = 5$ NIE jest lokalnym minimum. Wyszukujemy zatem we fragmencie $A[2 \ldots 2]$ odpowiadającym mniejszemu sąsiadowi, tj. 2.
- $L = 2$; $R = 2$; $M = (L+R)/2 = 2$. Element $A[2] = 2$ JEST lokalnym minimum. Koniec.

4. Wartość maksymalna w tablicy *bitonicznej*. Mówimy, że tablica jest bitoniczna (ang. bitonic) jeśli składa się z dwóch ciągów, z których pierwszy jest rosnącym ciągiem liczb całkowitych a drugi jest malejącym ciagiem liczb całkowitych. Przykład:

   - 0, 8, 18, 27, 16, 14, 5, -6, -15, -26.
   - 0, 3, 8, 16, 18, 8,-8,-25,-32,-45,-58,-62,-71,-88,-99, -107.

```
public static int[] bitonic(int N)
{
    Random ran = new Random();
    int mid = ran.nextInt(N);
    int[] a = new int[N];
    for (int i = 1; i < mid; i++) {
        a[i] = a[i-1] + 1 + ran.nextInt(9);
    }
    if (mid > 0) a[mid] = a[mid-1] + ran.nextInt(10) - 5;
    for (int i = mid + 1; i < N; i++) {
        a[i] = a[i-1] - 1 - ran.nextInt(9);
    }
    for (int i = 0; i < N; i++) {
        System.out.println(a[i]);
    }
    return a;
}
```

   Zaprojektuj i zaimplementuj algorytm, który, biorąc pod uwagę bitoniczny układ N różnych wartości całkowitych w tablicy, wyszuka element maksymalny w tej tablicy.

   Program (algorytm) powinien użyć $\sim log_2(N)$ porównań w najgorszym przypadku.

5. Wyszukiwanie *bitoniczne.* Dana jest tablica, która ma bitoniczny układ N różnych wartości całkowitych. Zaprojektuj i zaimplementuj algorytm, który sprawdza czy dana liczba całkowita key jest w tablicy.

   Program (algorytm) powinien użyć $\sim 3log_2(N)$ porównań w najgorszym przypadku.

   Podpowiedź: Użyj wersję wyszukiwania binarnego, aby znaleźć maksimum (w czasie $\sim log_2(N)$); następnie zastosuj wyszukiwanie binarne do przeszukania w powstałych kawałkach tablicy (każdy kawałek sprawdzamy w czasie $\sim log_2(N)$).